Playground Series - Season 3, Episode 2

Tabular Classification with a Stroke Prediction Dataset.

Prepared By : Soumee Ghosh (emp id: 2211444)

Dataset Description:

The dataset for this competition (both train and test) was generated from a deep learning model trained on the Stroke Prediction Dataset. Feature distributions are close to, but not the same, as the original. Feel free to use the original dataset as part of this competition, both to explore differences as well as to see whether incorporating the original in training improves model performance.

This dataset is a tabular classification dataset for stroke prediction. It is a binary classification problem, where prediction would be either a person had Brain stroke(1) or not(0). The features and their description are as follows:

- id: unique identifier
- gender: "Male", "Female" or "Other"
- age: age of the patient
- hypertension: 0 if the patient doesn't have hypertension, 1 if the patient has hypertension
- heart_disease: 0 if the patient doesn't have any heart diseases, 1 if the patient has a heart disease
- ever_married: "No" or "Yes"
- work_type: "children", "Govt_jov", "Never_worked", "Private" or "Self-employed"
- Residence_type: "Rural" or "Urban"
- avg_glucose_level: average glucose level in blood
- bmi: body mass index
- smoking_status: "formerly smoked", "never smoked", "smokes" or "Unknown"*
- stroke: 1 if the patient had a stroke or 0 if not.

The analysis can be divided into the following steps:

- Understanding the data
- Statistical analysis
- EDA
- Data pre-processing
- Model building
- Model Evaluation

a) Understanding the dataset:

Firstly, we load the training, testing and original dataset. Then we perform merging of training data with original data. Since both data are similar one is original and one synthetic, so we merge this data with our training data. It is called Data Augmentation, increasing the training data will help train the model better, the more the data, the better.

```
df = pd.concat([train_df, original_df]).reset_index(drop=True)
```

```
print("shape of training data before merging:", train_df.shape)
print("shape of training data after merging:",df.shape)
```

```
shape of training data before merging: (15304, 12)
shape of training data after merging: (20414, 12)
```

So we observe that post merging the number of observations increases from 15304 to 20414.

We then observe the datatypes of the features through the .info method.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20414 entries, 0 to 20413
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   id                 20414 non-null  int64  
 1   gender             20414 non-null  object 
 2   age                20414 non-null  float64
 3   hypertension       20414 non-null  int64  
 4   heart_disease      20414 non-null  int64  
 5   ever_married       20414 non-null  object 
 6   work_type          20414 non-null  object 
 7   Residence_type     20414 non-null  object 
 8   avg_glucose_level  20414 non-null  float64
 9   bmi                20213 non-null  float64
 10  smoking_status     20414 non-null  object 
 11  stroke             20414 non-null  int64  
dtypes: float64(3), int64(4), object(5)
memory usage: 1.9+ MB
```

So we observe that 3 features i.e. age, avg_glucose_level and bmi belong to the float type whereas id is integer and all other features belong to the object type. We can also see that there are no null values present except in the bmi feature.

In [12]:
```
df.isnull().sum()
```

Out[12]:
```
gender                 0
age                    0
hypertension           0
heart_disease          0
ever_married           0
work_type              0
Residence_type         0
avg_glucose_level      0
bmi                  201
smoking_status         0
stroke                 0
dtype: int64
```

b) Statistical analysis

We now look at the statistical description of the dataset.

```
]: df.describe()
```

]:

	id	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke
count	20414.000000	20414.000000	20414.000000	20414.000000	20414.000000	20213.000000	20414.000000
mean	14877.273636	41.870510	0.061673	0.031008	93.322256	28.302280	0.043157
std	16825.306948	21.756482	0.240567	0.173344	32.476351	7.021765	0.203215
min	0.000000	0.080000	0.000000	0.000000	55.120000	10.300000	0.000000
25%	4766.250000	25.000000	0.000000	0.000000	75.220000	23.500000	0.000000
50%	9511.500000	43.000000	0.000000	0.000000	86.250000	27.700000	0.000000
75%	14279.750000	58.000000	0.000000	0.000000	99.730000	32.200000	0.000000
max	72940.000000	82.000000	1.000000	1.000000	271.740000	97.600000	1.000000

We can observe that the mean age is around 42 years and the average glucose level is 93.22 whereas average bmi is 28.3.

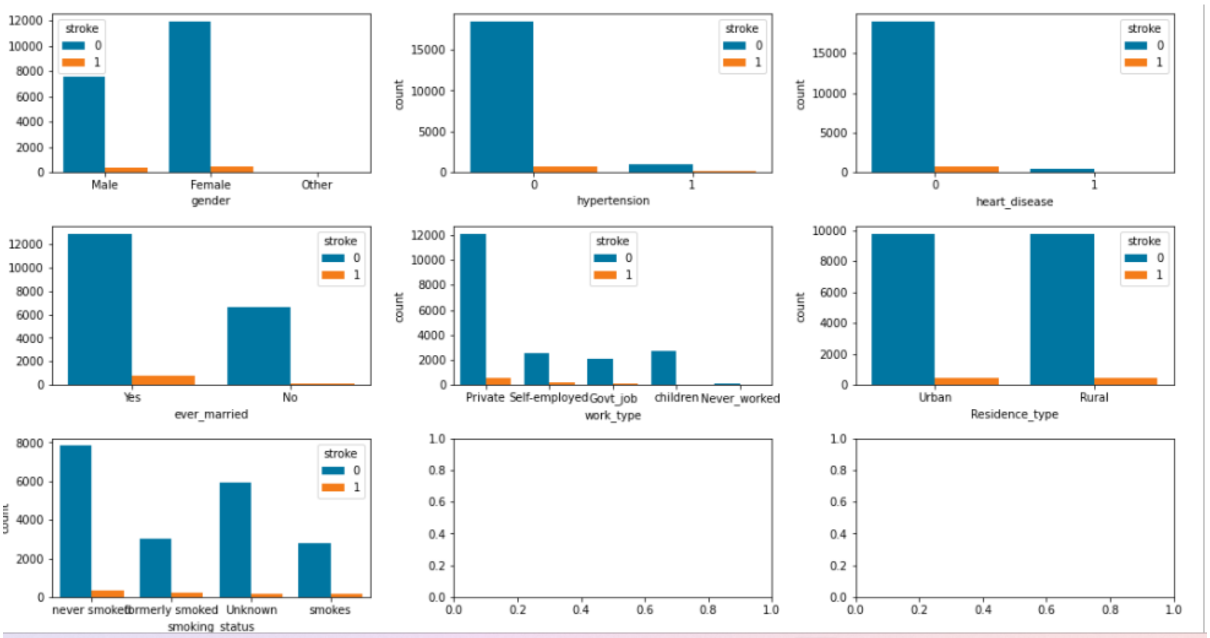
c) Exploratory Data Analysis

Next we try to divide our features into numerical and categorical features for the ease of our analysis.

We find out the no of unique categories for each feature using .nunique() method.

For the categorical features , we find the value counts of each category within a feature using count plot and for numerical features we check out their distribution using distplot.

Next we try to compare the stroke rate with respect to the categorical columns.



Observations:

Interestingly, the ratio of getting stroke is same whether patient is from rural and urban.

Patients who ever-married, are getting more strokes than non-married.

Ration of getting stroke or not is same for Private and self-employed people.

If patient is non-smoker, he or she have less chance of getting stroke and their ratio from barely smoker is almost same.

d) Data Pre-Processing

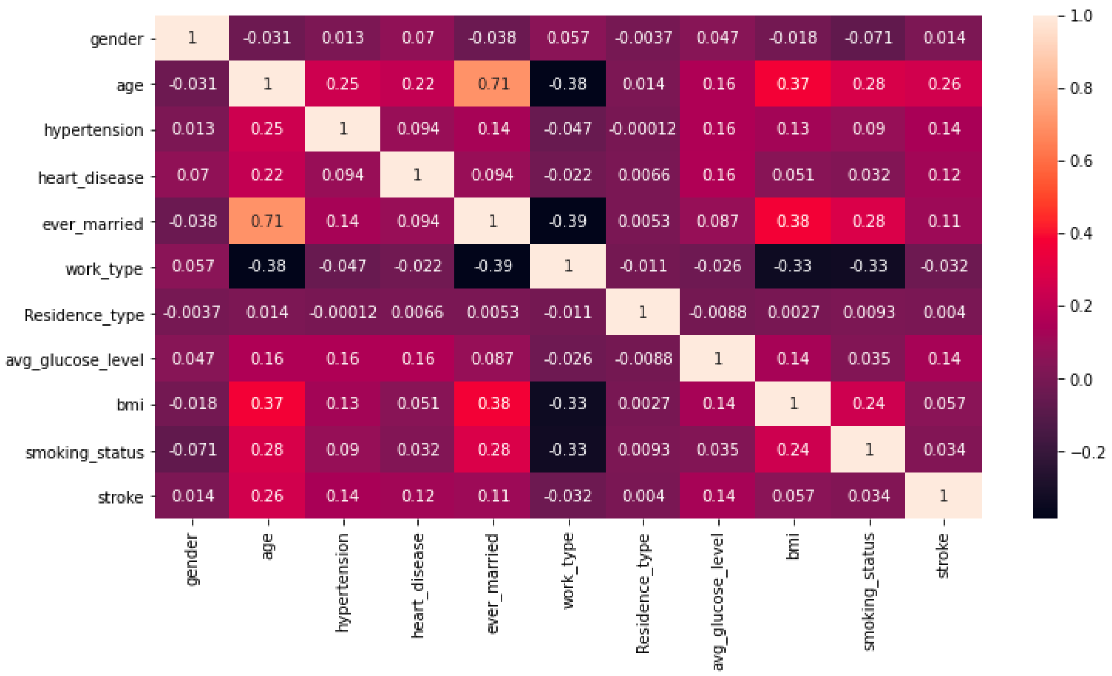
Thereafter we move to data pre-processing.

Firstly in order to deal with null values for the bmi feature we perform simple imputingby importing SimpleImputer from the sklearn libarray impute class. With the help of it we replace the null values with the median value.

For the categorical columns we perform encoding and using ordinal encoder we encode them.

Then we move to the bivariate analysis and using correlation we aim to look at the association between the variables.

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d3b0651070>
```

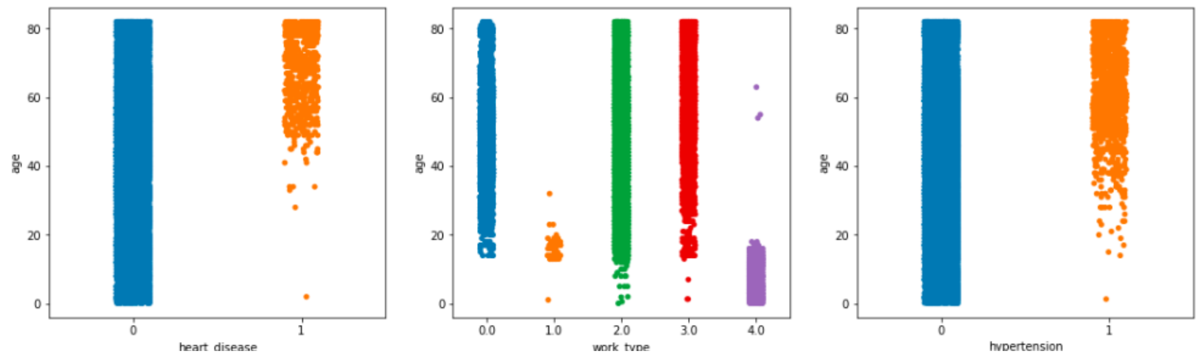


* Age and Worktype have strong negatinve correlation.

* Worktype and bmi have fairly good negative correlation

We then try to observe the relationship between Age vs heart disease, age vs work_type and age vs Hypertension.

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d3b044f520>
```



Through this strip plot we can conclude that:

* Almost all heart disease people are above 50, which is obvious.

* Hypertension disease in people of above 50.

Thereafter we split the features into X and y i.e. where y is our target feature i.e. stroke and X contains all other features.

We then scale the features using Maxmin scaler and split the data into training and validation set.

e) Model Building

Now we move to the model building section. Here we use the following models and check their performance score to choose the optimal model for our data.

- Logistic Regression
- Decision Tree Classifier
- Random Forest Classifier
- Neural Networks

Logistic Regression:

At first we build a logistic regression model and performing hyperparameter tuning with the help of Stratified K Fold ,cross validation score and Grid Search CV we tune our model to find the best parameters and best score.

```
Fitting 5 folds for each of 18 candidates, totalling 90 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   21.0s
[Parallel(n_jobs=-1)]: Done  90 out of  90 | elapsed:   22.7s finished
```

```
Best Score:  0.9583932070542129
Best Hyperparameters:  {'C': 1, 'penalty': 'l1', 'solver': 'liblinear'}
```

Decision Tree Classifier:

Similarly for decision trees we find

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    1.4s
Best Score:  0.8633707196785189
Best Hyperparameters:  {'max_depth': 5, 'min_samples_leaf': 20}
[Parallel(n_jobs=-1)]: Done  80 out of  80 | elapsed:    1.8s finished
```

Random Forest Classifier:

Now after performing hyperparameter tuning for random forest we obtained the following:

```
Fitting 5 folds for each of 80 candidates, totalling 400 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   32.8s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:  3.1min
[Parallel(n_jobs=-1)]: Done 400 out of 400 | elapsed:  7.2min finished
OOB SCORE : 0.9583932070542129
```

Finally training our model on neural networks gives us an average accuracy of 95.95 %.

f) Model Evaluation:

Finally we evaluate our models on the validation set and compare their scores:

Model	Logistic Regression	Decision Tree Classifier	Random forest Classifier	Neural Networks
score	95.18%	92%	95.12%	95.19%

So we can conclude that Logistic regression and neural networks model perform the best on our data.